Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Maks Peteh

FPGA simulator igre Gomoku

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Branko Šter

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Ker se pri Drevesnem preiskovanju Monte Carlo (Monte Carlo Tree Search) tipično izvaja veliko število naključnih simulacij, nas zanima, kako bi jih pohitrili s pomočjo strojnega izvajanja. Zapišite VHDL kodo za strojno implementacijo simulatorja za igro Gomoku. Ocenite, kolikšno pohitritev bi dobili, če vzamemo eno od pogosto uporabljanih programirljivih vezij FPGA.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Maks Peteh, z vpisno številko 63090145, sem avtor diplomskega dela z naslovom:

FPGA simulator igre Gomoku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Branka Štera,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 25. februarja 2016 Podpis avtorja:

Zahvaljujem se staršem za podporo in vzpodbudo med študijem, pa tudi mentorju za pomoč in potrpežljivost.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
VHDL	Very high speed integrated circuit hardware description language	Opisni jezik za digitalna integrirana vezija
MCTS	Monte Carlo tree search	Drevesno preiskovanje Monte Carlo
FPGA	Field-programmable gate array	Vrsta programljivih logičnih vezij
RTL	Register-transfer level	Registrsko-prenosni nivo
LFSR	Linear feedback shift register	Linearno-pomikalni register s povratno zanko

Povzetek

Naslov: FPGA simulator igre Gomoku

Drevesno preiskovanje Monte Carlo (Monte Carlo tree search, MCTS) je preiskovalna metoda, ki združuje natančnost drevesnega preiskovanja z naključnim vzorčenjem. Pri drevesno preiskovalni Monte Carlo metodi je treba izvesti veliko število simulacij v čim krajšem času. Ideja je hitro izvajanje velikega števila iger, ki bi na FPGA tekla hitreje kot na računalniškem procesorju. Cilj je razviti simulacijo igre Gomoku z igralcema, ki naključno izbirata poteze. V okviru diplomskega dela je bil razvit simulator, ki izbira poteze igralcev, jih zapiše v pomnilnik ter preveri zmagovalca. Za implementacijo je bil uporabljen jezik za opisovanje strojne opreme VHDL in XILINX urejevalnik, ter vmesnik za testiranje.

Ključne besede: VHDL, Gomoku, Artix 7.

Abstract

Title: Gomoku game FPGA simulator

Monte Carlo tree search is a search method that combines the precision of tree search with random sampling. In the method Monte Carlo tree search is to be carried out a large number of simulations in the shortest possible time. The idea is to implement a large number of games that would FPGA run faster than on a computer processor. The aim was to develop a simulation game Gomoku with players who randomly choose moves. In the context of the thesis has been developed simulator that chooses pleyers moves which are written in the memory and verification of the winner. For the implementation was used the language to describe hardware, VHDL and XILINX editor and interface for testing.

Keywords: VHDL, Gomoku, Artix 7.

Poglavje 1

Uvod

Z razvojem računalništva se vse bolj razvija tudi umetna inteligenca. Ena od preiskovalnih metod je drevesno preiskovanje Monte Carlo. Monte-Carlo simulacije vsebujejo zaporedja naključnih akcij. Prvič so bile uporabljene v računalnikih Los Alamos leta 1985. Danes imajo Monte Carlo simulacije vpliv na mnogih področjih, v kemiji, biologiji, ekonomiji, financah, itd. Drevesno-iskalni problemi so problemi, ki se lahko obdelujejo s postopno širitvijo v drevesno strukturo, na primer v iskanju poti in šahu. Drevesno preiskovanje Monte Carlo je splošni algoritem, ki se lahko uporablja za reševanje različnih problemov. Najbolj obetavni rezultati so pri igranju igre Go, kjer je bil daleč najbolj uspešen iskalni algoritem[3].

Cilj diplomske naloge je razviti simulator na FPGA, ki izvaja igro Gomoku, za dva igralca z naključnimi potezami. Vzporedno želimo preveriti, če igra teče hitreje kot na računalniku. Ideja je bila, da se namesto preverjanja zmagovalca v zanki, preverja zmagovalca vzporedno preko seštevalnikov. Igralca izbirata poteze izmenično, dokler ne dobimo zmagovalca, ali pa je izid neodločen.

V naslednjih poglavjih si bomo pogledali komponente simulatorja in tehnologije, ki so bile uporabljene. V prvem poglavju je predstavljen jezik VHDL,

ki je bil osnova za pisanje naloge, opisana je igra Gomoku in na kratko predstavljen njen potek. Za potek igre je bil potreben generator naključnih števil, uporabljen je bil LFSR. Pregledali bomo uporabljene knjižnice za definicijo signalov v komponentah in pomnilnika, kjer so shranjene poteze igralcev. Nato so predstavljene komponente krmilnika. Komponenta Ram služi hranjenju potez igralcev in seštevanju polj. Poleg tega komponenta pomaga pri določanju zmagovalca in preverja, če je igra končana. Komponenta Random določa naključne naslove s pomočjo LFSR, katere pošlje v Ram. V poglavju Testiranje si bomo pogledali primer testne datoteke in statistiko iger.

1.1 FPGA

Programabilne logične tehnologije, kot so FPGA, so osnovne komponente vsakega modernega načrtovalca vezij. Zaradi njihove razširitvene možnosti, ki so enolično prilagojene širokemu delu aplikacij, so FPGA idealni za rešitve mnogih problemov v hitro razvijajočem se tehnološkem sektorju. Glavne značilnosti programabilnih vezij so prilagodljivost, cenovna ugodnost in hitrejša izvedba zaradi paralelizma strojne opreme. Na trgu imamo različne ponudbe FPGA. Najbolj popularne so Artix-7, Kintex-7, Spartan-6, ... V našem primeru smo izbrali čip XC7A100T, ki je del serije Artix-7.

1.2 Artix 7

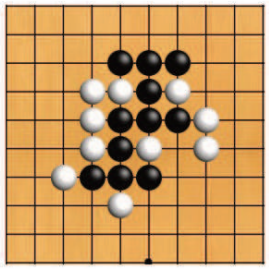
Naprave Artix-7 zagotavljajo najnižjo porabo in ceno pri 28nm tehnologiji. Optimizirani so tako, da nudijo najboljše delovanje v nizko-cenovni družini FPGA. Naprava Artix-7 je cenovno najugodnejša za aplikacije, pri katerih je pomembna nizka poraba. Vključuje programsko opredeljen radio, kamero s strojnim vidom in sistem brezžičnega prenosa[1].

Poglavje 2

Pojmi v simulatorju

2.1 Gomoku

Gomoku je družabna igra, imenovana tudi Omok ali 5 v vrsto. Gomoku igrata 2 igralca na polju velikosti 15 x 15. Igralec postavlja svoje figure na prazna mesta tako, da poskuša postaviti 5 enakih figur v vrsto. Ko figuro postavi, je ne sme premakniti. Primer igre je prikazan na sliki 2.1. V našem primeru smo vzeli polje 7 x 7 zaradi lažjega preverjanja zmagovalca igre.



Slika 2.1: Primer igre Gomoku

2.2 LFSR

Linearno-pomikalni register s povratno zanko (Linear feedback shift register, LFSR) je pomikalni register, pri katerem je vhod linearna funkcija XOR prejšnjega stanja. Delovanje opišemo z značilnim polinomom po modulu 2, vendar kot koeficienta nastopata le 0 in 1. LFSR je bil uporabljen za generiranje naključnih števil.

2.3 Bločni pomnilnik

Je pomnilnik, ki vsebuje več elementov enakega tipa. V našem primeru je pisanje sinhrono in branje asinhrono, ker potrebujemo več podatkov naenkrat. V simulatorju se pomnilnik uporablja za hranjenje potez igralcev. Definiran je z velikostjo naslova ter tipa podatkov, ki so shranjeni v njem.

2.4 VHDL

VHDL je kratica, ki pomeni standard za opisovanje, modeliranje in sintezo digitalnih vezij in sistemov.
Vsak opis je sestavljen iz treh delov. Prvi del so knjižnice, v katerih so definirani podatkovni tipi, funkcije, itd. Drugi del so entitete, ki vsebujejo definicijo prilkjučkov vezij. Tretji del pa so arhitekture, ki opisujejo delovanje vezja na tri načine. Prvi način je vzporedno, kar pomeni, da se stavki izvajajo paralelno. Drugi je zaporedno, kjer stavki se izvajajo zaporedno v procesih. Tretji pa je modularno, kjer uporabimo vnaprej sprogramirane komponente. Preslikava med priključki v VHDL kodi in fizičnimi priključki FPGA je opisana v UCF datoteki[5].

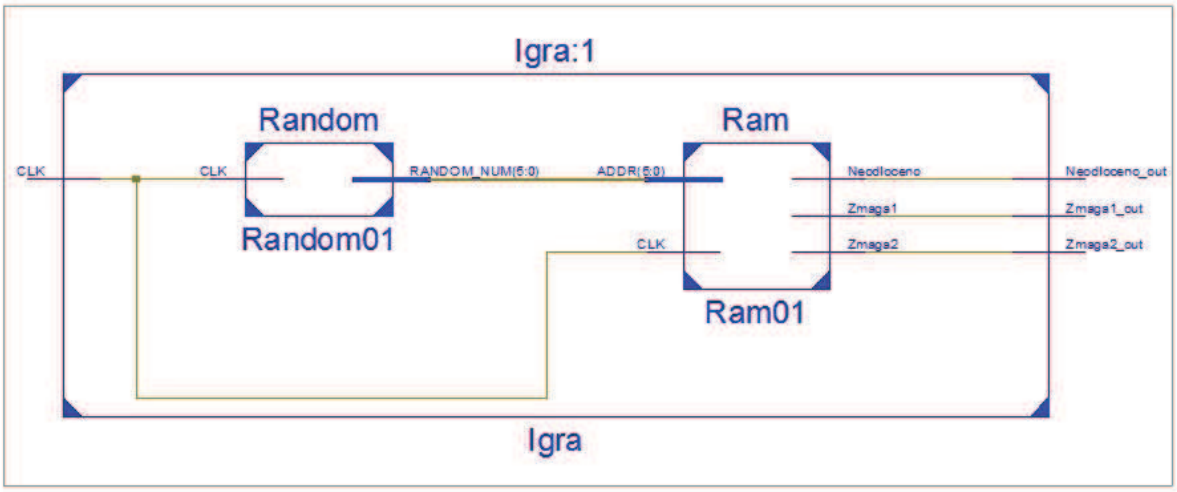
2.5 Knjižnice

Standard IEEE 1164 definira enote, ki vsebujejo deklaracijo in podpirajo enotno predstavitev logične vrednosti v VHDL opisu. Sponzorirana je bila s strani Design Automation Standards Committee od Institute of Electrical and Electronics Engineers. Standardizacija je bila osnovana na tipu deklaracije Synopsys MVL-9.

V okviru diplomske naloge smo uporabili standardne knjižnice, ki so definirane v IEEE standardu. Std_logic_1164 definira standard za opisovanje povezavnih podatkovnih tipov v VHDL. Tako kot Std_logic_1164, rabimo tudi std_logic_unsigned, ki je razširitev std_logic_arith, za delo s pozitivnimi števili[2].

2.6 RTL sheme

V načrtovanju digitalnih vezij je RTL abstrakcija načrta, ki ponazarja sinhrono digitalno vezje v smislu pretoka signala med hardverskimi registri in logičnimi operatorji, ki so izvedeni nad temi signali, kot je prikazano na sliki 2.2. RTL abstrakcija je uporabljena v opisnih jezikih strojne opreme, kot sta Verilog in VHDL. V shemi je prikazano vezje na višji ravni, iz katerega se lahko predstavi nižjo raven in na koncu dejansko ožičenje[4].



Slika 2.2: Primer sheme RTL

Poglavje 3

Simulator igre Gomoku

Igra poteka v določenem zaporedju, po katerem so komponente razdeljene. Simulacijo začnemo z izbiro naključnega števila v komponenti Random, katero pošlje v komponento Ram, kjer se zapiše v pomnilnik. Iz pomnilnika se števila prenesejo v komponente Adder. Po tem ko seštejemo vektorje v pomnilniku, se seštevki pošljejo v komponento Checker. Simulacija se izvaja v istem zaporedju, dokler ne dobimo zmagovalca, ali pa je polje polno.

3.1 Komponenta Random

Komponenta Random služi izbiranju naključnih naslovov v pomnilniku, kamor igralec zapiše svojo potezo. Naslov se izbere s pomočjo pomikalnega registra. Čeprav rabimo za naslov le 6 bitov, je pomikalni register 31 biten zaradi večje periode. Te bite dobimo s pomočjo funkcije XNOR, ki je negacija funkcije XOR. XNOR ima visok signal v primeru, kadar sta vhodna signala enaka. Večjo periodo rabimo, ker je generator psevdonaključen in se vrednosti po nekaj časa začnejo ponavljati. Komponenta ima le en vhodni in en izhodni signal. Vhodni CLK, ki je tipa STD_LOGIC, služi sinhronizaciji z uro. Izhodni RANDOM_NUM, tipa STD_LOGIC_VECTOR, velikosti 6 bitov izberemo iz vektorja Q. Prvih 6 bitov vektorja Q je naključni naslov. Preden se naključno število pošlje v naslednjo komponento, se preveri, če je

število med 0 in 48, kolikor je veliko igralno polje.

```
entity Random is
  port (
    CLK : in STD_LOGIC;
    RANDOM_NUM : out STD_LOGIC_VECTOR(5 downto 0) -- Address
  );
end Random;

architecture Behavioral of Random is

begin
  process (CLK)
  variable Q : STD_LOGIC_VECTOR (1 to 31):=
  (others => '0');
  variable ADDR : STD_LOGIC_VECTOR (5 downto 0) := "000000";
  begin
    if (rising_edge(CLK)) then
      Q(1) := Q(25) xnor Q(26);
      Q(2) := Q(26) xnor Q(27);
      Q(3) := Q(27) xnor Q(28);
      Q(4) := Q(28) xnor Q(29);
      Q(5) := Q(29) xnor Q(30);
      Q(6) := Q(30) xnor Q(31);
      Q(7 to 31) := Q(1 to 25);
    end if;
    ADDR := Q(1 to 6);
    if (ADDR <= "110000") then
        RANDOM_NUM <= ADDR;
    end if;
  end process;

end Behavioral;
```

3.2 Komponenta Ram

Komponenta Ram je razdeljena na več delov. Ima nalogo, da zapiše igralčeve poteze in preveri zmagovalca oziroma preveri, če je polje polno. Vrednosti v pomnilniku so shranjene kot 4 bitni vektorji, ki so na začetku "0000". Ta vrednost pomeni, da je polje prazno. Preden zapiše igralčevo potezo, preveri, če je polje prazno. Če polje ni prazno, se poteza ignorira in čaka na naslednji naključni naslov. Vsakič, ko se v polje uspešno zapiše vrednost, prištejemo 1 tej vrednosti, ki šteje število polnih polj.

```
process (ADDR,CLK, Zmaga1, Zmaga2, Neodloceno)
    variable RAM_ADDR_IN : integer range 0 to 48;
    variable STARTUP : boolean := true;
    begin
      if (STARTUP = true) then
        RAM256 <= (others =>"0000");
        POLNO_SIG_SUM <= "000000";
      end if;
      STARTUP := false;
      if (rising_edge(CLK)) then
        RAM_ADDR_IN := conv_integer(ADDR);
        if (RAM256(RAM_ADDR_IN) = "0000") then
          if (IZB = 1) then
            IZB <= 0;
          else
            IZB <= IZB + 1;
          end if;
          RAM256(RAM_ADDR_IN) <= IZBIRA(IZB);
          POLNO_SIG_SUM <= POLNO_SIG_SUM + '1';
        end if;
      end if;
      if (Zmaga1 = '1' or Zmaga2 = '1' or Neodloceno = '1') then
        STARTUP := true;
      end if;
  end process;
  NUM <= RAM256(0 to 48);
```

3.3 Komponenta Adder

Poteza prvega igralca je vrednosti ”0001”, drugega pa ”0010”. Seštevalnik služi seštevanju vrednosti v pomnilniku. Sešteje 5 števil, kolikor je potrebnih za zmago, vendar le če so vsa števila večja od ”0000”. Imamo 3 vrste seštevalnikov zaradi polj različnih velikosti. Komponenta dobi vrednosti iz pomnilnika, ki so tipa STD_LOGIC_VECTOR velikosti štirih bitov in jih sešteje. Dobi rezultat, ki je vrednosti med ”0101”in ”1010”, prav tako tipa STD_LOGIC_VECTOR velikosti štirih bitov.

3.3.1 Adder7

Adder7 se uporablja v primeru, kjer je vektor velikosti 7. Sešteje števila od NUM1 do NUM5, od NUM2 do NUM6 in od NUM3 do NUM7. Dobi 3 vsote, SUM1, SUM2 in SUM3, ter jih pošlje v komponento, kjer se preverja zmagovalca. Primer seštevkov je prikazan na sliki 3.1, kjer seštejemo polja 1, 8, 15, 22, 29, 36, 43, ki so primer prvega vektorja in 21, 22, 23, 24, 25, 26, 27, ki so primer drugega.

```
entity Adder7 is
  Port( NUM1 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM2 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM3 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM4 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM5 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM6 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM7 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM1 : out STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM2 : out STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM3 : out STD_LOGIC_VECTOR(3 downto 0) := "0000"
      );
end Adder7;

architecture Behavioral of Adder7 is
begin
    SUM1 <= NUM1 + NUM2 + NUM3 + NUM4 + NUM5 when (NUM1 > "0000")
```

```
    and (NUM2 > "0000") and (NUM3 > "0000")
    and (NUM4 > "0000") and (NUM5 > "0000")
    else "0000";
    SUM2 <= NUM2 + NUM3 + NUM4 + NUM5 + NUM6 when (NUM2 > "0000")
    and (NUM3 > "0000") and (NUM4 > "0000")
    and (NUM5 > "0000") and (NUM6 > "0000")
    else "0000";
    SUM3 <= NUM3 + NUM4 + NUM5 + NUM6 + NUM7 when (NUM3 > "0000")
    and (NUM4 > "0000") and (NUM5 > "0000")
    and (NUM6 > "0000") and (NUM7 > "0000")
    else "0000";
end Behavioral;
```

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

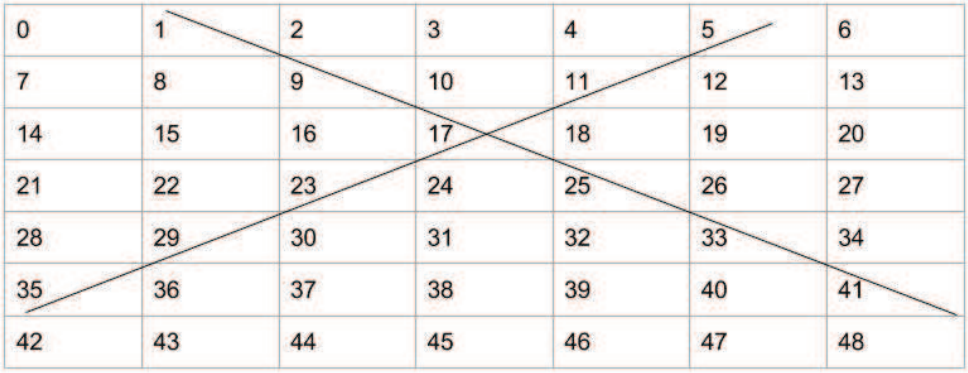
Slika 3.1: Primer vektorjev dolžine 7

3.3.2 Adder6

Adder6 se uporablja v primeru, kjer je vektor velikosti 6. Sešteje števila od NUM1 do NUM5 in od NUM2 do NUM6. Vrne vsoti SUM1 in SUM2, ki se pošljeta v naslednjo komponento. Primer seštevkov je na sliki 3.2, kjer seštejemo polja 1, 9, 17, 25, 33, 41, ki so primer prvega vektorja in 5, 11, 17, 23, 29, 35, ki so primer drugega.

```
entity Adder6 is
  Port( NUM1 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM2 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM3 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

```
      NUM4 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM5 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      NUM6 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM1 : out STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM2 : out STD_LOGIC_VECTOR(3 downto 0) := "0000"
      );

end Adder6;

architecture Behavioral of Adder6 is

begin
  SUM1 <= NUM1 + NUM2 + NUM3 + NUM4 + NUM5 when (NUM1 > "0000")
   and (NUM2 > "0000") and (NUM3 > "0000")
   and (NUM4 > "0000") and (NUM5 > "0000")
  else "0000";
  SUM2 <= NUM2 + NUM3 + NUM4 + NUM5 + NUM6 when (NUM2 > "0000")
   and (NUM3 > "0000") and (NUM4 > "0000")
   and (NUM5 > "0000") and (NUM6 > "0000")
  else "0000";

end Behavioral;
```



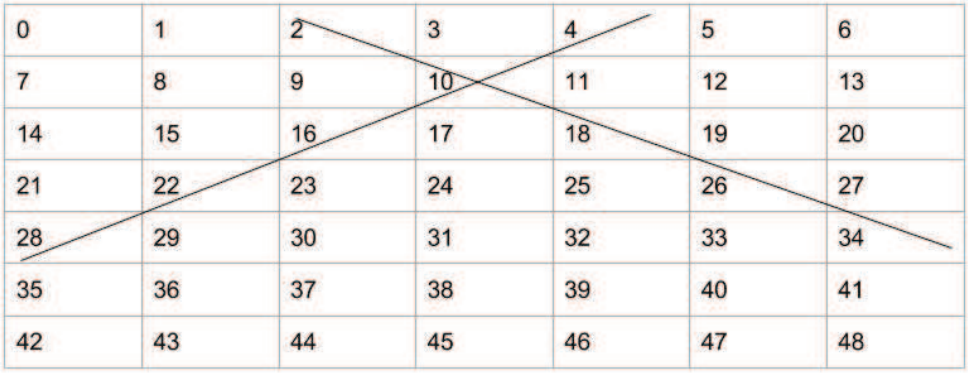
Slika 3.2: Primer vektorjev dolžine 6

3.3.3 Adder5

Adder 5 se uporablja, kadar je vektor velikosti 5. Sešteje vsa števila, NUM1 do NUM5 in dobljeno SUM1 pošlje v naslednjo komponento. Na sliki 3.3 je primer seštevkov vektorja z naslovi polj 2, 10, 18, 26, 34 in vektorja 4, 10, 16, 22, 28.

```
entity Adder5 is
  Port(NUM1 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
       NUM2 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
       NUM3 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
       NUM4 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

```
      NUM5 : in STD_LOGIC_VECTOR(3 downto 0) := "0000";
      SUM1 : out STD_LOGIC_VECTOR(3 downto 0) := "0000"
    );

end Adder5;

architecture Behavioral of Adder5 is

begin
  SUM1 <= NUM1 + NUM2 + NUM3 + NUM4 + NUM5 when (NUM1 > "0000")
  and (NUM2 > "0000") and (NUM3 > "0000")
  and (NUM4 > "0000") and (NUM5 > "0000")
  else "0000";

end Behavioral;
```



Slika 3.3: Primer vektorjev dolžine 5

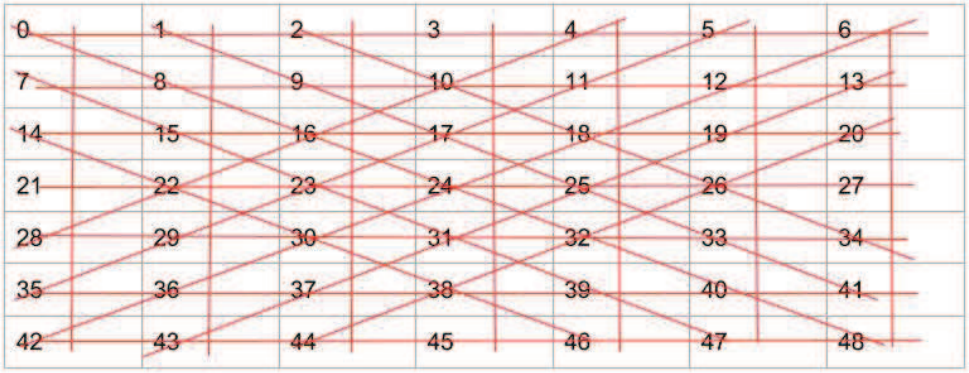
3.4 Komponenta Checker

Ko seštejemo vsa polja, lahko preverimo zmagovalca. Komponenta Checker dobi vrednosti seštevalnikov, ki so vse možnosti vektorjev dolžine 5,6 ali 7 v polju. Vseh seštevkov je 60, ki se shranijo v vektor. Na sliki 3.4 so vse možnosti polj, ki jih komponenta preveri. Komponenta ima dva vhodna signala. Prvi je SUM, tipa Matrika1, katerega smo definirali sami. Je vektor velikosti 60, treh bitnih STD_LOGIC_VECTOR. Drugi Polno_sum je 5 bitni vektor, seštevek polnih polj. Skozi zanko v procesu preverjamo, če je katera od vrednosti v SUM "0101", kar pomeni prvega zmagovalca ali "1010", kar

pomeni drugega zmagovalca. Če je vrednosti Polno_sum večja od 49, je izid neodločen. Če dobimo zmagovalca ali je izid neodločen, se vrednosti v komponenti Ram ponastavijo na "0000".

```
PACKAGE matrika_pkg IS
    type Matrika1 is array (1 to 60) of STD_LOGIC_VECTOR(3 downto 0);
END;

entity Checker is
  port (
  SUM : in Matrika1;
  Zmaga1 : buffer STD_LOGIC;
  Zmaga2 : buffer STD_LOGIC;
  Neodloceno : buffer STD_LOGIC;
  Polno_sum : in STD_LOGIC_VECTOR(5 downto 0)
  );
end Checker;

architecture Behavioral of Checker is
signal Polno : STD_LOGIC;
begin
  Polno <= '1' when Polno_sum >= "110000" else '0';
  process(SUM)
  begin
    Zmaga1 <= '0';
    for i in 1 to 60 loop
      if (SUM(i) = "0101") then
        Zmaga1 <= '1';
      end if;
    end loop;
  end process;

  process(SUM)
  begin
    Zmaga2 <= '0';
    for i in 1 to 60 loop
      if (SUM(i) = "1010") then
        Zmaga2 <= '1';
```

```
      end if;
    end loop;
  end process;
  Neodloceno <=  '1' when Polno = '1' and Zmaga1 = '0' and Zmaga2 = '0'
  else '0';

end Behavioral;
```



Slika 3.4: Polja, kjer so možnosti za zmago so označena z rdečo

Poglavje 4

Testiranje

Testiranje smo simulirali na čipu XC7A100T, v simulatorju XILINX. Funkcije omogočajo izračun najvišje frekvence in najmanjšo periodo. Izvaja se s pomočjo datoteke test bench, v kateri se definira signale za simulacijo.

4.1 Test bench

Test bench je pogosto navidezno okolje, ki je uporabljeno za preverbo pravilnosti ali stabilnosti načrta ali modela. Izraz ima korenine pri testiranju elektronskih naprav v laboratoriju, kjer je inženir izvajal teste z različnimi pripravami za merjenje in manipulacijo, kot so osciloskop, klešče, multimeter, itd. V kontekstu inženiringa, se test bench nanaša na okolje, v katerim s pomočjo orodij testiramo produkt v razvoju. Testiranje je pogosto zasnovano za točno določen produkt. V našem primeru nastavimo začetno vrednost ure na '0' in določimo njeno periodo. Vzeli smo najmanjšo možno urino periodo 10 nanosekund, pri kateri program še pravilno teče.

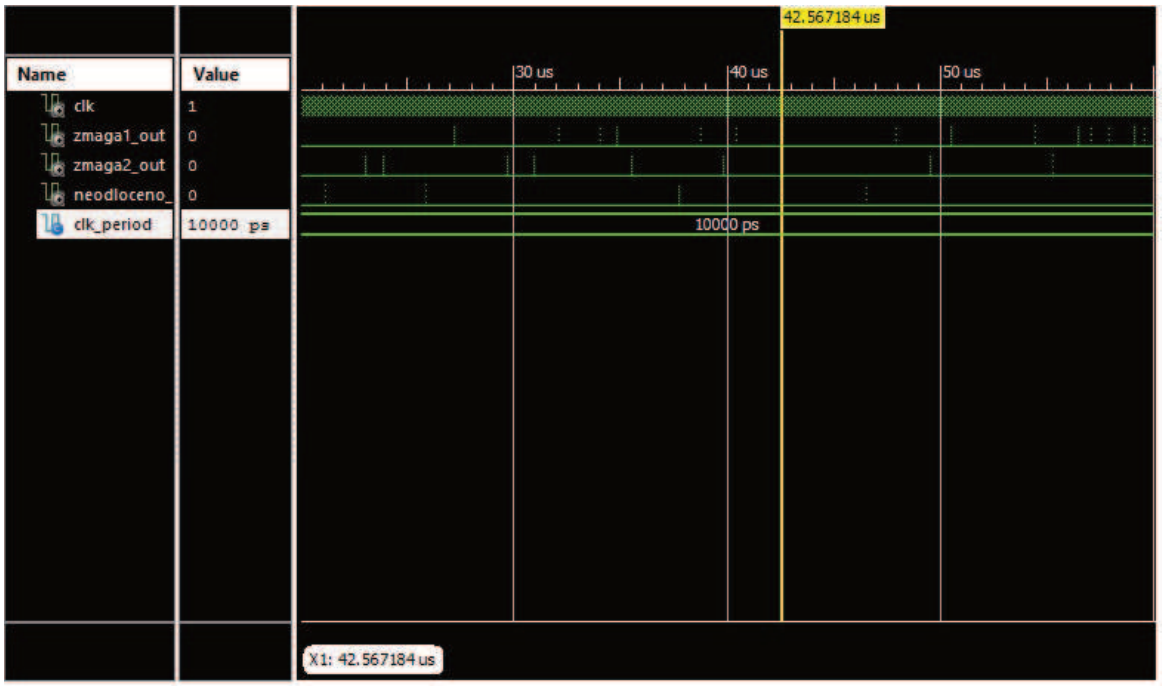
```
ENTITY Igra_tb IS
END Igra_tb;

ARCHITECTURE behavior OF Igra_tb IS

    COMPONENT Igra
```

```
    PORT(
         CLK : IN  std_logic;
         Zmaga1_out : buffer  std_logic;
         Zmaga2_out : buffer  std_logic;
         Neodloceno_out : buffer  std_logic
        );
    END COMPONENT;

   signal CLK : std_logic := '0';

   signal Zmaga1_out : std_logic;
   signal Zmaga2_out : std_logic;
   signal Neodloceno_out : std_logic;

   constant CLK_period : time := 10 ns;

BEGIN

   uut: Igra PORT MAP (
          CLK => CLK,
          Zmaga1_out => Zmaga1_out,
          Zmaga2_out => Zmaga2_out,
          Neodloceno_out => Neodloceno_out
        );

   CLK_process :process
   begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
   end process;

   stim_proc: process
   begin

   end process;
```

```
END;
```

4.2 Primer igre

Pred začetkom igre lahko nastavimo seme generatorja naključnih števil. Igro poženemo v simulatorju XILINX in jo lahko izvajamo poljubno dolgo. Na sliki 4.1 je prikazan primer simulacije. Prvi signal prikazuje urino periodo, drugi in tretji signal sta igralca, četrti pa je neodločen izid. Ko dobimo zmagovalca ali neodločen izid, je tisti signal visoke napetosti pol urine periode.



Slika 4.1: Primer igre

4.3 Statistika iger

Najdaljši čas poteze igralca z izbiranjem naključnega števila je bil malo manj kot 10ns, kar pomeni, da je najvišja frekvenca čipa približno 100MHz. Povprečen čas igre je bil približno 1900 nanosekund. V idealnem času bi bil povprečen čas igre 240 nanosekund, vendar je daljši zaradi izbiranja polja v Ram. Poteza se izbira naključno, dokler ne najde praznega polja. Kljub temu je čas poteze približno 40-krat hitrejši kot na računalniškem procesorju, čas igre pa v povprečju 10-krat hitrejši. Ker je izbiranje potez naključno, je število zmag prvega in drugega igralca približno enako. Večino manj je neodločenih izidov, kot je zmag prvega ali drugega igralca.

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge smo uspešno razvili simulator, ki simulira igro Gomoku, ter jo koristili za izvajanje igre na FPGA simulatorju. Testiranje implementacije je po pričakovanjih tekla razmeroma hitro. Narejena implementacija se je izkazala za veliko hitrejšo od implementacije na procesorju v večini primerov. Lahko bi tekla hitreje, če bi imeli boljši sistem za določanje naključnih števil in boljši način preverjanja, če je naslov že poln. Ena od idej je bila, da se naslovi premešajo v vektorski tabeli in se nato zapisujejo v pomnilnik. S tem bi določili večje najmanjše število potez igre, vendar ne bi imeli problemov pri podvajanju izbranih potez za igralca. Diplomsko delo se lahko uporablja za raziskovanje metode drevesnega preiskovanja Monte Carlo in pa tudi v drugih metodah, kjer je potrebno hitro simuliranje naključnosti. Lahko se uporablja napisane komponente tudi kot modul pri drugih simulatorjih.

Literatura

[1] Artix-7. [Online]. Dosegljivo: http://www.xilinx.com/products/silicon-devices/fpga/artix-7.html [Dostopano 3. 3. 2016].

[2] Knjižnice. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/IEEE_1164 [Dostopano 3. 3. 2016].

[3] MCTS. [Online]. Dosegljivo: https://project.dke.maastrichtuniversity.nl/games/files/phd/Chaslot_thesis.pdf. [Dostopano 29. 2. 2016].

[4] RTL. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Register-transfer_level [Dostopano 2. 3. 2016].

[5] VHDL. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/VHDL [Dostopano 3. 3. 2016].